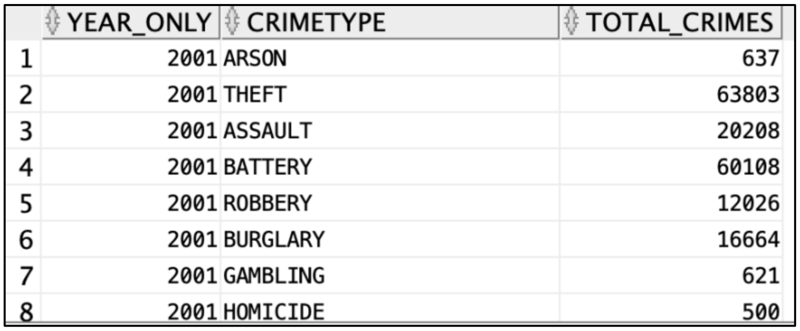
P2 Assignment: Analytic SQL

ISM 6208 Data Warehousing
Dalton Anderson, Kevin Hitt

Query 1: Aggregations with CUBE and ROLLUP

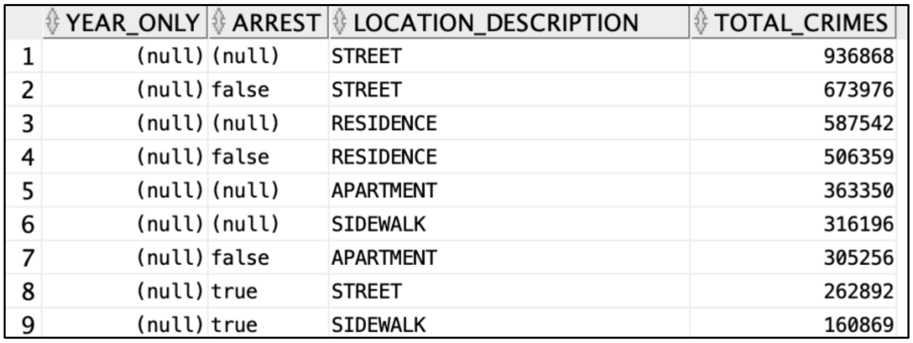
The use of ROLLUP and CUBE here allows us to analyze crime data at different levels of aggregation. ROLLUP provides subtotals and grand totals for crimes by type and year, while CUBE goes further by considering all possible combinations. By utilizing ROLLUP, we might be able to see patterns/trends otherwise unnoticeable without the aggregation. This provides a useful tool for understanding the relationships between crime types and their occurrences over different years.

```
-- Use ROLLUP to get aggregate of crimes by type by year:
SELECT YEAR_ONLY, CRIMETYPE, COUNT(*) as TOTAL_CRIMES
FROM PEARL_CHICAGO
GROUP BY ROLLUP (YEAR_ONLY, CRIMETYPE);
```



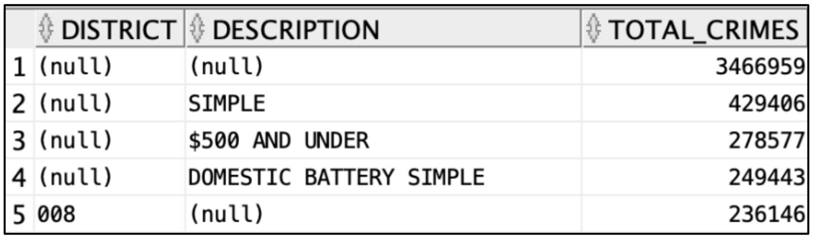
	YEAR_ONLY	CRIMETYPE	TOTAL_CRIMES
1	2001	ARSON	637
2	2001	THEFT	63803
3	2001	ASSAULT	20208
4	2001	BATTERY	60108
5	2001	ROBBERY	12026
6	2001	BURGLARY	16664
7	2001	GAMBLING	621
8	2001	HOMICIDE	500

```
-- USE CUBE to get crimes per year per arrest status,
-- as well as the total number of crimes per location description
SELECT YEAR_ONLY, ARREST, LOCATION_DESCRIPTION, COUNT(*) as TOTAL_CRIMES
FROM PEARL_CHICAGO
GROUP BY CUBE (YEAR_ONLY, ARREST), LOCATION_DESCRIPTION
ORDER BY TOTAL_CRIMES DESC;
```



	YEAR_ONLY	ARREST	LOCATION_DESCRIPTION	TOTAL_CRIMES
1	(null)	(null)	STREET	936868
2	(null)	false	STREET	673976
3	(null)	(null)	RESIDENCE	587542
4	(null)	false	RESIDENCE	506359
5	(null)	(null)	APARTMENT	363350
6	(null)	(null)	SIDEWALK	316196
7	(null)	false	APARTMENT	305256
8	(null)	true	STREET	262892
9	(null)	true	SIDEWALK	160869

```
-- Use both ROLLUP and CUBE to get the total crimes per district,
-- per description of the crime, and across all descriptions and districts:
SELECT DISTRICT, DESCRIPTION, COUNT(*) as TOTAL_CRIMES
FROM PEARL_CHICAGO
GROUP BY ROLLUP (DISTRICT), CUBE (DESCRIPTION)
ORDER BY TOTAL_CRIMES DESC;
```



	DISTRICT	DESCRIPTION	TOTAL_CRIMES
1	(null)	(null)	3466959
2	(null)	SIMPLE	429406
3	(null)	$500 AND UNDER	278577
4	(null)	DOMESTIC BATTERY SIMPLE	249443
5	008	(null)	236146

Query 2: Computing RANKs

While the sparseness of the Chicago Crime data doesn’t lend itself to an obvious comparison of RANK v. DENSE_RANK, here we will consider a common student assessment example. We can compute the RANK of each student based on their scores. The student with the highest score will have a RANK of 1; the next highest will have a RANK of 2, and so on.

```
SELECT student_name, score,
    RANK() OVER (ORDER BY score DESC) AS rank,
    DENSE_RANK() OVER (ORDER BY score DESC) AS dense_rank
FROM student_scores;
```

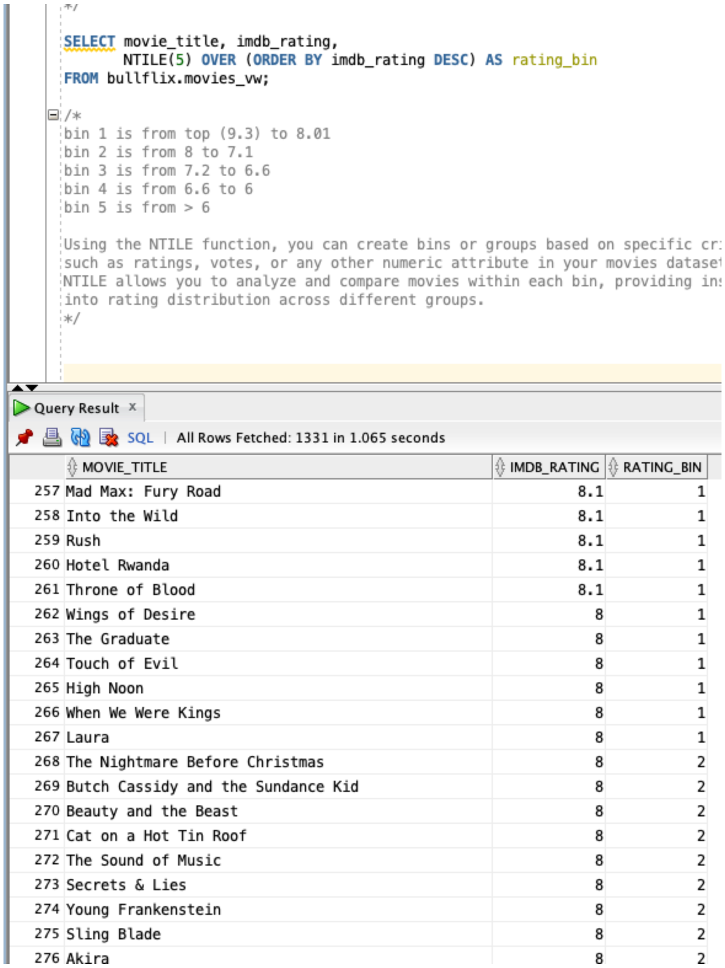
```
student_name | score | rank | dense_rank
-------------*-------*------*-----------
John         | 95    | 1    | 1
Emily        | 90    | 2    | 2
David        | 88    | 3    | 3
Sophia       | 88    | 3    | 3
Michael      | 85    | 5    | 4
```

The main difference here between RANK() and DENSE_RANK() is how they handle ties (cases where multiple rows have the same value). The RANK() function leaves gaps in the ranking sequence when there are ties [Sophia -> Michael], while the DENSE_RANK() function assigns consecutive ranks to tied rows without any gaps [Sophia -> Michael].

Query 3: Creating Bins with NTILE

Creating bins with NTILE is a technique in SQL that allows you to divide a dataset into equal-sized groups or compartments based on a specified criterion. It is beneficial for analyzing data distribution and performing data segmentation. For example, let's consider a table of products with their prices. We can use NTILE to create bins or groups of products based on their price range. This can help identify price segments or categories for further analysis or marketing strategies.

Here's an example query that creates bins using NTILE:



```
*/

SELECT movie_title, imdb_rating,
       NTILE(5) OVER (ORDER BY imdb_rating DESC) AS rating_bin
FROM bullflix.movies_vw;

/*
bin 1 is from top (9.3) to 8.01
bin 2 is from 8 to 7.1
bin 3 is from 7.2 to 6.6
bin 4 is from 6.6 to 6
bin 5 is from > 6

Using the NTILE function, you can create bins or groups based on specific cr
such as ratings, votes, or any other numeric attribute in your movies dataset
NTILE allows you to analyze and compare movies within each bin, providing ins
into rating distribution across different groups.
*/
```

	MOVIE_TITLE	IMDB_RATING	RATING_BIN
257	Mad Max: Fury Road	8.1	1
258	Into the Wild	8.1	1
259	Rush	8.1	1
260	Hotel Rwanda	8.1	1
261	Throne of Blood	8.1	1
262	Wings of Desire	8	1
263	The Graduate	8	1
264	Touch of Evil	8	1
265	High Noon	8	1
266	When We Were Kings	8	1
267	Laura	8	1
268	The Nightmare Before Christmas	8	2
269	Butch Cassidy and the Sundance Kid	8	2
270	Beauty and the Beast	8	2
271	Cat on a Hot Tin Roof	8	2
272	The Sound of Music	8	2
273	Secrets & Lies	8	2
274	Young Frankenstein	8	2
275	Sling Blade	8	2
276	Akira	8	2

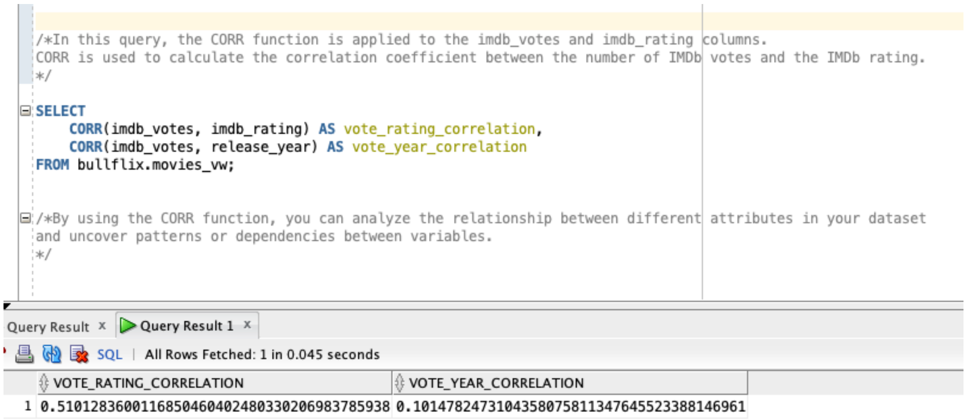
/*In this query, the NTILE function is applied to the imdb_rating column
The NTILE(5) specifies that the ratings will be divided into 5 equal-sized bins.
The ORDER BY imdb_rating DESC orders the movies based on their IMDb ratings in descending order.
*/

```
SELECT movie_title, imdb_rating,
       NTILE(5) OVER (ORDER BY imdb_rating DESC) AS rating_bin
FROM bullflix.movies_vw;

/*
bin 1 is from top (9.3) to 8.01
bin 2 is from 8 to 7.1
bin 3 is from 7.2 to 6.6
bin 4 is from 6.6 to 6
bin 5 is from > 6

Using the NTILE function, you can create bins or groups based on specific criteria,
such as ratings, votes, or any other numeric attribute in your dataset.
NTILE allows you to analyze and compare movies within each bin, providing insights across
different groups.
*/
```

Query 4: Correlations (CORR)

Correlations (CORR) in SQL is a powerful statistical function that allows you to measure the strength and direction of the linear relationship between two numeric variables in a dataset. It provides valuable insights into the dependency and association between variables. For example, consider a table with data on housing prices and square footage. Using the CORR function, we can determine the correlation coefficient between these two variables, which indicates how closely they are related.

Here's an example query that calculates the correlation coefficient:



```
/*In this query, the CORR function is applied to the imdb_votes and imdb_rating columns.
CORR is used to calculate the correlation coefficient between the number of IMDb votes and the IMDb rating.
*/

SELECT
    CORR(imdb_votes, imdb_rating) AS vote_rating_correlation,
    CORR(imdb_votes, release_year) AS vote_year_correlation
FROM bullflix.movies_vw;


/*By using the CORR function, you can analyze the relationship between different attributes in your dataset
and uncover patterns or dependencies between variables.
*/
```

```
All Rows Fetched: 1 in 0.045 seconds

  VOTE_RATING_CORRELATION                   VOTE_YEAR_CORRELATION
1 0.5101283600116850460402480330206983785938 0.1014782473104358075811347645523388146961
```

```
/*In this query, the CORR function is applied to the imdb_votes and imdb_rating columns.
```

```
CORR is used to calculate the correlation coefficient between the number of IMDb votes and
the IMDb rating.
*/



SELECT
   CORR(imdb_votes, imdb_rating) AS vote_rating_correlation,
   CORR(imdb_votes, release_year) AS vote_year_correlation
FROM bullflix.movies_vw;

/*By using the CORR function, you can analyze the relationship between different attributes in
your dataset
and uncover patterns or dependencies between variables.
*/
```

The results align with expectations where the year is a limited influence as movies are timeless. In comparison, the amount of votes a movie receives is moderately correlated with the rating outcome of the film. For a studio, an important question is whether X person is worth the money. A studio can look at the correlation between Quentin Tarantino and his movies.